Enseirb, filière informatique, 2ème année

Projet de Programmation Multicœur et GPU
Travail en binôme

Jeu de la vie

Le jeu de la vie (cf. https://fr.wikipedia.org/wiki/Jeu_de_la_vie) inventé par J. Conway en 1970, simule une forme très basique de vie en partant de deux principes simples : pour pouvoir vivre, il faut avoir des voisins ; mais quand il y en a trop, on étouffe. Le *monde* est ici un grand damier de cellules vivantes ou mortes, chaque cellule étant entourée par huit voisines. Pour faire évoluer le monde on découpe le temps en étapes et, pour passer d'une étape à la suivante, on compte pour chaque cellule le nombre de cellules vivantes parmi ses huit voisines puis on applique les règles suivantes :

- Une cellule morte devient vivante si elle a exactement 3 cellules voisines vivantes – autrement elle reste morte.
- Une cellule vivante reste vivante si elle est entourée de 2 ou 3 cellules vivantes – autrement elle meurt.

Notre objectif est de comparer l'efficacité différentes implémentations de ce jeu. En particulier nous allons, dans un premier temps, produire trois versions séquentielles :

- Version de base : le monde est parcouru à l'aide de deux boucles *for* imbriquées ;
- Version tuilée : on considère que le monde est divisé en tuiles carrées (tuiles de 32 x 32 cellules, par exemple) - le monde est parcouru selon les tuiles (quatre boucles *for* imbriquées) ;
- Version optimisée : le monde est parcouru selon les tuiles mais on évite de recalculer les tuiles que l'on sait stagnantes c'est-à-dire que l'on sait qu'elles ne vont pas évoluer lors de la prochaine itération.

Puis il s'agira de paralléliser ces versions à l'aide d'OpenMP (omp for, omp task) et OpenCL. Voici les versions attendues :

- Versions OpenMP for :
  - Version de base
  - Version tuilée (on pourra utiliser la directive collapse pour distribuer les tuiles)
  - Version optimisée
- Versions OpenMP task :
  - Version tuilée
  - Version optimisée
- Versions OpenCL
  - Version de base (la plus naïve possible)
  - Version optimisée

Enfin, on développera une version mixte OpenCL + OpenMP en répartissant une partie du monde sur CPU et l'autre partie sur CPU et le GPU.

Cas à étudier sur les machines de la salle 203 ou le serveur tesla pour OpenMP et les salles 201 & 202 pour OpenCL

Il s'agit de comparer l'efficacité des différentes techniques de parallélisation en produisant des courbes de *speed-up* fonction du nombre de threads employés (pour OpenMP) et la taille des tuiles (pour OpenCL). Pour ce faire, on considèrera les paramètres suivants :

- Motif initial de la configuration : « guns » ou « random ».
- Taille du tableau : 256, 1024, 4096
- Nombre d'itérations : 100, 1000, 10000
- Type de scheduling : static, dynamic
- Taille des tuiles : 16 & 32 pour OpenCL et 32, 64 & 128 pour OpenMP

L'objectif est donc de comparer les performances des techniques employées sur différents cas mais aussi d'expliquer autant que possible le comportement du programme. Il s'agit de décrire les courbes obtenues, de les comparer et de repérer les phénomènes étranges. Ensuite, il s'agit de raisonner à partir de la description des courbes et de formuler des hypothèses pour expliquer les comportements observés et, idéalement, de valider / infirmer les hypothèses à l'aide de nouvelles expériences. Exemples d'observations et de pistes d'interprétations :

- Un speedup supérieur au nombre de cœurs est constaté. Avez-vous utilisé le meilleur algorithme séquentiel ? Gagne-t-on grâce au non-déterminisme du parallélisme ? Bénéficie-t-on d'effets de cache favorables ?

- On observe que l'ordonnancement "dynamic" est beaucoup moins performant que le "static". Est-ce un problème de contention sur un mutex ? Est-ce un problème de localité mémoire (cache / NUMA) ?

- On constate que l'ordonnancement "static" est beaucoup moins performant que le "dynamic". La charge de travail est-elle bien équilibrée ? la mémoire est-elle bien répartie sur la machine ?

- On observe que la courbe d'accélération croit puis décroit. Y'a-t-il suffisamment de travail pour compenser le surcoût de la parallélisation ? Y-a-t-il un déséquilibre dû à *l'hyper-threading* ? Y a-t-il de la contention (cache / mémoire) ?

L'idée n'est pas de produire toutes les courbes possibles mais de proposer des comparaisons judicieuses mettant en relief les différentes techniques programmées.

Éléments mis à votre disposition

Un programme calculant un stencil (ici la transposée) à partir des cas tests est mis à votre disposition. L'option --help permet de visualiser l'ensemble des options accessibles en ligne de commande. L'option « -d –p » permet de faire une pause entre chaque itération. Pour la mesure des performances on utilisera l'option « -n » - on pourra également visualiser la vitesse instantanée du calcul via l'option « -d t ». De plus, l'interface utilisateur, via les touches up et down, permet d'afficher les configurations périodiquement (une configuration sur dix par exemple). Vous pouvez enrichir cette interface afin de faire varier le nombre de threads utilisés ou encore de basculer interactivement d'une technique de calcul. Un script R et un script shell vous sont également fourni afin éditer des courbes de *speed-up*.

Rendu

Rapport et codes sources seront à déposer dans un répertoire au CREMI qui vous sera précisé ultérieurement. Le rapport présentera des extraits du code illustrant les différentes techniques employées, les expériences réalisées et leurs commentaires.